

CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science Department

Silicon Structures Project

SSP File #4332

R L A P

Version 1.0

A Chip Assembly Tool

by

R.C. Mosteller

June 1981

Copyright, California Institute of Technology, 1981

Table of Contents

1. INTRODUCTION	1
1.1 The Cell	2
1.2 The Connectors and Ports	3
1.2.1 Ports	3
1.2.2 Connectors	4
1.3 Optimized CIF	5
2. Using Rlap	6
2.1 Getting Started	6
2.2 Reference Pointers	6
2.3 Creating a Cell	7
2.4 Creating Instances	8
2.5 Setting the Current Layer	10
2.6 Creating Ports	10
2.7 Creating Wires	10
2.8 Creating Boxes	11
2.9 Creating Circles	11
2.10 Creating Text	11
3. Using External Cells	12
4. River Router Procedures	13
5. Defining a Technology Dependent Part	14
6. NMOS Technology Dependent Part	17
6.1 NMOS Technology Code	17
6.2 An Example	21
7. Utility Programs Written in Rlap	24
7.1 CIF to CIF	24
7.2 Stick to CIF	24

List of Figures

Figure 1-1: Cell Model

2

1. INTRODUCTION

The RLAP system is a programming tool to generate integrated circuit geometry in a convenient manner. The model used is a rectangular cell with connection points called ports. Hierarchical design methodology is well supported within RLAP. The output from RLAP is an optimized CIF 2.0¹ file [MEAD 80] with user extensions for cell name² and ports³. External cells are supported via the Sticks Standard [Trimberger 80] and CIF 2.0 [MEAD 80].

RLAP is embedded in the SIMULA programming language [Birtwistle 73]. The full power of the extensive SIMULA programming language can be applied to the generation of chip geometry. SIMULA is a block structured language which requires all identifiers to be declared in the block header before use. It is not necessary for a user to have an extensive knowledge of SIMULA, however a knowledge of programming and an overview of SIMULA would ease the task of designing with RLAP.

RLAP is partitioned into two parts. The first is the data structures in the form of SIMULA classes and the routines for generation of geometry. The second is the technology dependent part. This part contains the valid layer names, wire widths, design rules and sticks models for a specific technology. Currently there is one model supported for NMOS. This model supports the rules defined in Mead and Conway [MEAD 80].

The user defines his or her primitive elements in terms of a scale factor LAMBDA [MEAD 80]. This allows easy changing of the design for various scalings. The current NMOS model uses LAMBDA of 2.5 microns.

¹ CIF stands for Caltech Intermediate Form.

² User extension 9 is used for cell names. example: 9: A CELL;

³ User extension 5 is used for ports. Example: 5: 10 20 "A Port";

1.1 The Cell

A cell is considered to be rectangular in shape as shown in figure 1-1. All geometrical data is on the interior of the rectangle and considered to be the property of the cell. The cell contains the physical data to realize the cell. The ports of the cell are the interface to the outside world. Each port defines a position on the perimeter of the cell where a connection is to be accomplished. These ports protrude from the boundary of the cell like fingers.

Each port of the cell has a name, location, layer, side and wire width information. The cell may be interrogated for its ports.

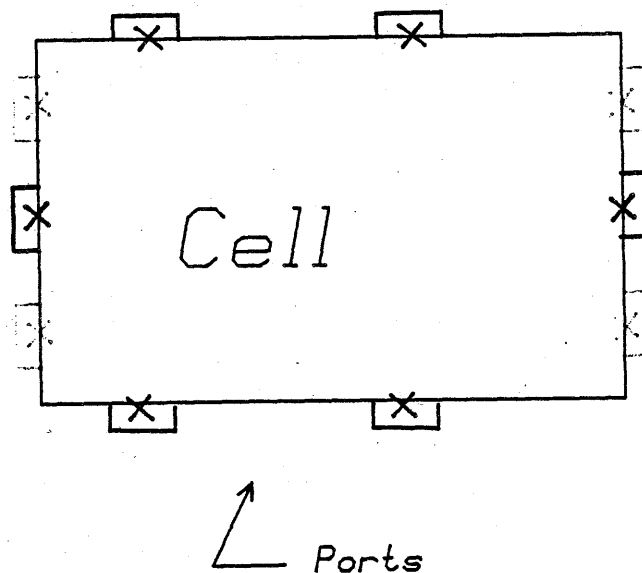


Figure 1-1: Cell Model

The dimensions of the cell are defined by two type of rectangles. The Cellmbb is the minimum bounding box for the cell. This box includes the extensions of the ports and includes all geometry of the cell. The second is the Cellabut box. This box encloses the cell through the ports and extends beyond any portless side by one half of the maximum

design rule⁴. This box is used to place cells side by side with no design rule errors. In addition by placing cells juxtaposition, connections of the two cells is accomplished by abutment. That is, the ports of one cell are coincidence with the other. If the ports do not line up a river routing routine may be used, see section 4 page 13.

1.2 The Connectors and Ports

The ports and connectors provide the interface to the cell. Ports are individual connection points like pins on a chip. A connector is a set of ports.

1.2.1 Ports

A port defines a unique location on a specific layer. A port has a name that is a SIMULA text string. This name can be used in the look up process. The port is used as a connection point to a cell. That is, a wire can be run to a port on an instance of a cell to accomplish a connection. The port is a sub class of a point. That means that a port can be used everywhere a point can be used. The properties of a port may be accessed by "port.<property>". The following is a list of the accessible properties of ports.

X	This is the x value for the port. This is a real type.
Y	This is the y value for the port. This is a real type.
COLOR	This is the numerical number for the layer of the port. This is an integer type. In the technology part there is a set of predefined layer integers that can be used to test for layer. An example for NMOS is <u>p.color=red</u> .
MYNAME	This is the name of the port. This is a text string. All SIMULA tests on this parameter may be used.
WIDTH	This is the wire width for a connection to the port. This is of type real. The units are in LAMBDA.
SIDE	This is an integer representation of the side. RLAP has 5 variables declared for the side. they are <u>leftside</u> , <u>rightside</u> , <u>topside</u> , <u>bottomside</u> and <u>inside</u> . The meaning of the sides is intuitive in the

⁴For NMOS it is 1.5 LAMBDA. This parameter is settable in the technology dependent part see section 5 page 14.

names. To test whether a port is on a specific side a simple test may be used as p.side=leftside. This would be a test for the left side of a port.

1.2.2 Connectors

Each cell has a set of ports called connectors. The connector is a SIMULA class that holds a set of ports. The connector has a set of functions that can be used to extract ports or new connectors which contain subsets of the original connector. The functions are called by using the "connector.<function>" where function is the desired operation. The definition of these functions follow.

SETSIDE(side)	This function set all the ports in the connector to a specific side. The side is passed as a parameter.
CLEARSIDE	This function sets all of the sides in the connector to nil. This can be used in the definition of a cell which will allow the RLAP to generate the sides.
CLONE	This function creates a copy of the current connector. The type returned is connector.
VALPNT(i)	This function returns a port at index <u>i</u> .
VALX(i)	This function returns the x value which is real from the port at index <u>i</u> .
VALY(i)	This function returns the y value which is real from the port at index <u>i</u> .
VALLAYER(i)	This function returns the layer value which is an integer from the port at index <u>i</u> .
LEFT	This function returns a subconnector of those ports on the left side. The type is connector.
RIGHT	This function returns a subconnector of those ports on the right side. The type is connector.
TOP	This function returns a subconnector of those ports on the top side. The type is connector.
BOTTOM	This function returns a subconnector of those ports on the bottom side. The type is connector.
BYNAM(t)	This function returns the first port of name <u>t</u> .

- BYNAMCOLOR(t,c)** This function returns the first port of name t and color c.
- SELECT(t)** This function returns a new connector where the name matches t. The name is matched by wild card where * is the wild card character.
- SUB(i,j)** This function returns a subset of the connector where i is the starting index and j is the ending index.

1.3 Optimized CIF

The generated CIF from RLAP has been optimized for size and plotting speed. Each cell's data is kept in memory until it is completely defined. Then the cell data is sorted and written to the CIF disk file. The ports are sorted by color, side and location and written to disk first. Next the calls or instances are sorted by location and then written to disk. Finally the rest of the data is ordered by color and location, and then written to disk. The location is defined by the data type. All cells including external from stick or CIF files are optimized.

The wires are also checked at this time. To be written to disk a wire must have more than one point. Duplicate points are deleted.

The CIF file generated contains no forward references. Forward references in CIF files that are externally loaded will be resolved.

2. Using Rlap

2.1 Getting Started

To use RLAP, the user should use the following template. The users program is inserted at the appropriate place in the following template. The external class RNLAP is a version of RLAP for NMOS technology, see section 6 page 17. If you would like to define your own technology class see section 5 page 14.

```
BEGIN
EXTERNAL CLASS RNLAP5;
RNLAP
BEGIN

    ! put your Rlap code here;

END;
END OF PROGRAM;
```

2.2 Reference Pointers

In using RLAP in general it may be necessary to use references to various data types. It is only required to have one type of references for the various primitives. A primitive can reference an instance, a wire, a box, a flash, and a polygon. Cells can be referenced by a symbol. Points and ports can be referenced by a point. The following are examples of declarations of references for each.

```
ref(primitive) i,j,k;
ref(symbol) s,c;
ref(point) p,p1;
ref(port) pt;
ref(connector) c;
```

⁵The identifier RNLAP is the technology class for RLAP. For each different technology there would be a different class.

2.3 Creating a Cell

The definition of a symbol or cell is accomplished by the defined and enddef statement as shown below. Each cell should contain a recognizable structure. These structures may be called by the instances type statement. Various methods can be applied to generate ports for the cell. First the addport statement may be used. Or the addconnector from an instance may be used.

```
DEFINE("name of cell");
! cell body goes here;
ENDDF;
```

The enddef statement causes the cell to be written to the disk file. Any port that is defined to be on a side will either be snapped to its side or a wire added to move the port to the correct side. The decision is based on how the port was created. The enddef also returns a reference to the symbol.

The cell definitions may be nested if desired or if convenient. This allows creating a symbol for use in the cell instead of creating the actual geometry. The scope of a nested cell is global and may be used in other cells. This nesting facility is used in the creation of contacts and transistors in the readstick statement and by the technology dependent contacts.

A symbol reference may be set to a specific symbol by the <symbol reference>⁶
 :-CELL("symbol na
 symbol reference.

CONNECTORS	This function returns a connector for all the ports of the cell.
LEFT	This function returns a subconnector of those ports on the left side of the cell. The type is connector.
RIGHT	This function returns a subconnector of those ports on the right side of the cell. The type is connector.
TOP	This function returns a subconnector of those ports on the top side of the cell. The type is connector.

⁶The bracketing symbols <> generally mean that this is where the users data or identifiers go.

BOTTOM	This function returns a subconnector of those ports on the bottom side of the cell. The type is connector.
BYNAM(t)	This function returns the first port of name <u>t</u> of the cell.
BYNAMCOLOR(t,c)	This function returns the first port of name <u>t</u> and color <u>c</u> of the cell.
LENGTH	This function returns the length of the cell in LAMBDA.
WIDTH	This function returns the width of the cell in LAMBDA.
CELLMBB	This function returns a rectangle ⁷ for the minimum bounding box for the cell.
CELLABUT	This function returns a rectangle for the abutment box of the cell.

2.4 Creating Instances

Calls to symbols may be accomplished in several ways depending on the transformation to be applied. In addition information about connectors, ports, and location may be requested from the instances. This information will be correct for the applied transformations. That is, port sides, and locations will be adjusted according to the transformation.

An instance is created by a Callcell("symbol name") or Callref(<symbol reference>) expression. This expression returns an instance reference. Transformations are applied to this reference by the dot operator. For example to call a cell at location X 10 do callcell("dummy").x(10). The transformations may be concatenated. Each transformation returns a reference to the instance. The following is the set of transformations that may be used.

MIRRORX	This transformation causes the instances to be mirrored in x. That is the new value of x is equal to minus the old value of x.
MIRRORY	This transformation causes the instances to be mirrored in Y. That is the new value of y is equal to minus the old value of y.

⁷ In RLAP a rectangle is a class with two points ll for lower left and ur for upper right. To access the lower left hand x coordinate of the cell one would use: cellmbb.ll.x. In addition the other corners may be accessed by lr for lower right and ul for upper left.

ROTATE(a,b)	This function causes the instances to be rotated by the rotation vector a,b.
PNT(p)	This function causes the instances to be translated to the point p.
XY(x,y)	This function causes the instances to be translated to the location x,y.
X(x)	This function causes the instances to be translated to the position x by value x.
Y(y)	This function causes the instances to be translated to the position y by value y.
LOWERLEFT	This function causes the origin point to be translated to the lower left hand corner of the instances.
LOWERRIGHT	This function causes the origin point to be translated to the lower right hand corner of the instances.
UPPERLEFT	This function causes the origin point to be translated to the upper left hand corner of the instances.
UPPERRIGHT	This function causes the origin point to be translated to the upper right hand corner of the instances.

The following functions are used to get various data about the instance at the point of translation.

CONNECTORS	This function returns a connector for all the ports of the instance.
LEFT	This function returns a subconnector of those ports on the left side of the instance. The type is connector.
RIGHT	This function returns a subconnector of those ports on the right side of the instance. The type is connector.
TOP	This function returns a subconnector of those ports on the top side of the instance. The type is connector.
BOTTOM	This function returns a subconnector of those ports on the bottom side of the instance. The type is connector.
BYNAM(t)	This function returns the first port of name <u>t</u> of the instance.
BYNAMCOLOR(t,c)	This function returns the first port of name <u>t</u> and color <u>c</u> of the instance.

LENGTH	This function returns the length of the instance in LAMBDA.
WIDTH	This function returns the width of the instance in LAMBDA.
CELLMBB	This function returns a rectangle for the minimum bounding box for the instance.
CELLABUT	This function returns a rectangle for the abutment box of the instance.

2.5 Setting the Current Layer

The current layer may be set by the statement layer(l). Where l is the new default layer. Any primitive that requires a layer will use the default layer. If unspecified an error message will occur. The value names for the layers are defined in the technology part, see section 6 page 17 for the NMOS layers.

2.6 Creating Ports

A port can be created for the current cell by the statement addport(x,y,n,s). X and y are the location. N is the name of the port, an example would be "GND". S is the side. The identifiers for valid sides are defined in section 1.2.1 page 3. In addition there is an undefined side called nilside. Using this as a side tells RLAP to define the side.

Ports may be added from called instances by the addconnector(fa) statement. The term fa may be a port or connector.

2.7 Creating Wires

There are two types of expressions to generate wires. One for normal wires which is nwire. The other is Widewire(w) where w is the width of the wire. The path of the wire is defined by a series of dot operators and functions. The wire only extends beyond the end points by one half of the minimum width of the wire. This allows convenient connections for wires.

The following is a list of path functions for the wire.

X(x)	This function adds a new point at x and the prior points y.
Y(y)	This function adds a new point at y and the prior points x.

DX(x)	This function adds a new point at $x +$ prior point x and the prior points y .
DY(y)	This function adds a new point at $y +$ prior point y and the prior points x .
XY(x,y)	This function adds a new point at x,y .
DXY(x,y)	This function adds a new point at $x +$ the prior points x , $y +$ the prior points y .
PNT(p)	This function adds a new point at the point p .
LASTISPORT(n,s)	This function creates a port at the last point with name n and side s .

2.8 Creating Boxes

Boxes are created with the statement box(x1,y1,x2,y2). The point $x1,y1$ is the point representing the lower left hand corner of the box and $x2,y2$ is a point representing the upper right hand corner.

In addition to creating boxes by the box statement, a box can be created by the pad statement. A pad is a square box with a center point and a width. A pad is created by the pad statement pad(w,x,y). The w is the width, x and y is the location of the pad.

2.9 Creating Circles

A circle can be created by the statement circle(d,x,y). Where d is the diameter and x,y is the location.

2.10 Creating Text

Text can be created with the statement alpha(t,x,y,s). The text t is drawn to the right at location x,y . The scale factor multiplies all the dimensions of the generated geometry to allow the characters drawn to be enlarged. With a scale factor of 1, the size of the characters will be 9 by 7 lambda.

3. Using External Cells

There are two means to get external cells. One is from CIF files and the other is from stick files. The statement READCIF("CIF file") will cause the CIF file to be read and the symbols in that file to be added to the current set. Symbol numbers will be resolved at this time. In addition, ports and names will be extracted from the cell.

The statement READSTICK("stick file") will read the stick file symbols into the current set. The ports and names will be extracted from the cells.

The external cells are handled in a uniform manner and may be used as any other cell created in RLAP.

4. River Router Procedures

There are two river routing procedures in RLAP. The first is used to route a set of points while the second routes a set of ports with the appropriate wire widths and layers.

Riverfred(sep,offset,width,vb,vt)

This is the general river router for routing vectors of points. The sep is the separation between wires. The offsett is the offset from vector vb before routing can commence. Width is the wire width. Vb is the starting vector of points and vt is the final vector of points.

Riverport(sep,offset,vb,vt)

This is the port router. the sep is the wire separation. The offset is the offset from the connector vb. The vb is the starting connector and vt is the final connector.

5. Defining a Technology Dependent Part

The technology dependent part defines the specific attributes for a technology. The SIMULA model form is shown below. There are two parts to the model, one is the STICK model for stick standard and the other is the CIF parts. An INTEGER identifier should be defined for each layer for later use in the users defined cells.

```
SRASIM CLASS <Technology Name>
begin
  INTEGER <specific layer names>8;

  rcmodels CLASS models;
  BEGIN
    REF(symbol) PROCEDURE stickcell(l,w,token);REAL l,w;
    TEXT token;
    BEGIN
      TEXT t;
      REF(symbol) sym;
      INTEGER j;
      IF token=<Component type> THEN
        BEGIN
          t:-copy(<Component type>);
          sym:-cell(t);
          IF sym == NONE THEN
            BEGIN
              define(t);
              <component definition goes here>
              sym:-enddef;
            END;
          END ELSE
            blue:=metal:=0;
            ciflayer(0):-copy("NM");
            stklayer(0):-copy("METAL");
            wirewidth(0):=3;
            thespace(0):=metal;
            thespace(1):=metal;
            thespace(2):=3;
        END of class models;

        NEW models(<number of layers>,<size of space table>);

        lambda:=<lambda in microns>;
        max-lambda-rule:=<The maximum design rule in lambda>;
        cifscale:=<scale of output CIF cells>;
        snaptoedge:=<set to true is port defined on a side
          are to be snapped to the edged of the cell>;
```

⁸The bracketing <> defines a point where user data is to be supplied. The data inside the bracket defines the user type of data to be placed at that point.

terse:=<set to true for compacted cif file. No comments
or blank lines>;

end of model.

There are several global constants that are used in RLAP. Two are used for scaling and defining the A and B⁹ part in a cell definition[MEAD 80]. The following defines the various global constants in RLAP.

Lambda	This is the global unit of lambda. It is a real number in hundredth of microns.
Max+lambda+rule	This is the offset of a cell to define the cell abutment box. It should be one half of the maximum design rule.
Cifscale	This is the b part in CIF cells. This value is multiplied by every point on output of the cell. In a sense it is the rounding factor to integers. Cifscale is an integer.
Snaptoedge	If this boolean is set to true all ports that have an edged side defined will be snapped to that side. If the port is not on the end of a wire a new wire will be added from the port to the edge with a new port created.
Terse	When this is set the output CIF file will have no comments and all unnecessary blanks will be removed.

The following defines the various read only arrays for the technology. In general there is one entry for each layer in the array. The layers are numbered from 0 to N where N is the maximum number of layers. The value N in the following represents the current layer being defined. All of the arrays must be defined for the layers. If the value for a layer is not defined an error would occur for that layer. For instance wires can only be generated for layers which have a wire width.

Ciflayer(n)	This array defines the textual name for the CIF layer. It must be stored in capitalization and with the copy verb as: <u>:-copy("NM")</u> .
Stklayer(n)	This array defines the textual name for the STICK layer. It must be stored in capitalization and with the copy verb as:

⁹The a and b part are the division b and the multiplier a for a cell definition. Example:
DS 3 250 10 A corresponds to 250 and b to 10.

`:-copy("METAL").`

Wirewidth(n) This defines the LAMBDA wire width for layer N.

Thespace(X*3) This defines the spacing between two layers. The array contains three words for each space group. There are: 1) The space in LAMBDA, 2) The from layer, and 3) The to layer.

6. NMOS Technology Dependent Part

The following are the predefined layers for NMOS.

<u>NMOS layer</u>	<u>Accepted Names</u>
Metal	Metal, Blue
Polysilicon	Poly, Red
Diffusion	Diffusion, Green
Contact Cuts	Cuts, Black
Implant	Implant, Yellow
Over Glass	Glass
Buried	Buried

There are some built in facilities for generating contacts. They are as follows.

rb(x,y)	Red to blue contact centered at x,y
gb(x,y)	Green to blue contact
be(x,y)	Green to red butting contact, red facing east
bs(x,y)	Green to red butting contact, red facing south
bw(x,y)	Green to red butting contact, red facing west
bn(x,y)	Green to red butting contact, red facing north

The origin of the butting contact commands is on the centerline of the long axis of the symbol one lambda in from the green end.

6.1 NMOS Technology Code

The following is the actual code for the NMOS technology dependent parts. Notice the use of layer identifiers for indexes into the corresponding array.

In the Stick Cell definition, notice that they are defined with normal RLAP code and treated as a cell. Doing this allows using the normal code for processing stick files.

```
! nmos technology stuff;
INTEGER green,diffusion,red,poly,yellow,implant;
INTEGER cuts,contact,blue,metal,glass,black,buried;

rcmodels CLASS models;
BEGIN
  REF(symbol) PROCEDURE stickcell(l,w,token);REAL l,w;
  TEXT token;
  BEGIN
    TEXT t;
    REF(symbol) sym;
    INTEGER j;
    IF token="NRES" THEN
```

```

BEGIN
  t:-copy("RCMPULLTRANSL");
  t:-conc2(t,fastcvs(l));
  t:-conc2(t,"W");
  t:-conc2(t,fastcvs(W));
  sym:-cell(t);
  IF sym == NONE THEN
    BEGIN
      define(t);
      ! el butto;
      layer(green); box(-2,-3,2,1);
      addport(0,-2,"DSOURCE",inside);
      layer(red); box(2,0,-2,3);
      addport(0,+2,"PSOURCE",inside);
      layer(cuts); box(-1,-2,1,2);
      layer(metal); box(-2,-3,2,3);
      addport(0,-1,"MSOURCE",inside);

      layer(red);
      box(-w/2-2,0,w/2+2,1+L);
      layer(green);
      box(-w/2,1,w/2,1+L+2);
      addport(0,+2+L,"DRAIN",inside);
      layer(implant);
      box(-w/2-1.5,-1.5,w/2+1.5,1+L+1.5);
      sym:-enddef;
    END;
  END ELSE
  IF token="NENH" OR token="NDEP" THEN
    BEGIN
      j:=IF token="NENH" THEN 1 ELSE 2;
      IF j=1 THEN
        t:-copy("RCMENHTRANSL") ELSE
        t:-copy("RCMDEPTRANSL");
        t:-conc2(t,fastcvs(l));
        t:-conc2(t,"W");
        t:-conc2(t,fastcvs(W));
        sym:-cell(t);
        IF sym == NONE THEN
          BEGIN
            define(t);
            layer(red);
            box(-w/2-2,-l/2,+w/2+2,l/2);
            addport(0,0,"G1",inside);
            !addport(+1,0,"G2",inside);
            layer(green);
            box(-w/2,-l/2-2,+w/2,l/2+2);
            addport(0,-2,"SOURCE",inside);
            addport(0,2,"DRAIN",inside);
            IF j=2 THEN
              BEGIN
                layer(implant);
                box(-L/2-1.5,-w/2-1.5,+L/2+1.5,+w/2+1.5);
              END;
            sym:-enddef;
          END;
        END;
      END ELSE

```

```

IF token="NBUT" THEN
BEGIN
  t:-copy("RCMNBUT");
  sym:-cell(t);
  IF sym == NONE THEN
  BEGIN
    define(t);
    ! el butto;
    layer(green); box(-2,-3,2,1);
    addport(0,-2,"D",inside);
    layer(red); box(2,0,-2,3);
    addport(0,+2,"P",inside);
    layer(cuts); box(-1,-2,1,2);
    layer(metal); box(-2,-3,2,3);
    addport(0,-1,"M",inside);
    sym:-enddef;
  END;
END ELSE
IF token="NBUR" THEN
BEGIN
  t:-copy("RCMNBUR");
  sym:-cell(t);
  IF sym == NONE THEN
  BEGIN
    define(t);
    layer(red);
    box(-L/2,-w/2-2,+L/2,+w/2+2);
    addport(0,0,NOTEXT,inside);
    layer(green);
    addport(0,0,NOTEXT,inside);
    box(-L/2-2,-w/2,+L/2+2,+w/2);
    IF j=2 THEN
    BEGIN
      layer(implant);
      box(-L/2-1.5,-w/2-1.5,+L/2+1.5,+w/2+1.5);
    END;
    sym:-enddef;
  END;
END ELSE
IF token="NPM" THEN
BEGIN
  t:-copy("RCMNRD");
  sym:-cell(t);
  IF sym == NONE THEN
  BEGIN
    define("RCMNRD");
    layer(red); pads(4,0,0);
    addport(0,0,NOTEXT,inside);
    layer(cuts); pads(2,0,0);
    layer(metal); pads(4,0,0);
    addport(0,0,NOTEXT,inside);
    sym:-enddef;
  END;
END ELSE
IF token="NDM" THEN
BEGIN
  t:-copy("RCMNGD");

```

```

sym:-cell(t);
IF sym == NONE THEN
BEGIN
  define("RCMNGD");
  layer(green); pads(4,0,0);
  addport(0,0,NOTEXT,inside);
  layer(cuts); pads(2,0,0);
  layer(metal); pads(4,0,0);
  addport(0,0,NOTEXT,inside);
  sym:-enddef;
END;
END ELSE
IF token="NCON" THEN
BEGIN
  t:-copy("RCMNCON");
  sym:-cell(t);
  IF sym == NONE THEN
  BEGIN
    define("RCMNCON");
    layer(metal); pads(4,0,0);
    addport(0,0,NOTEXT,inside);
    layer(green);addport(0,0,NOTEXT,inside);
    layer(red);addport(0,0,NOTEXT,inside);
    sym:-enddef;
  END;
  END;
  stickcell:-sym;
END of stickcell;

blue:=metal:=0;
ciflayer(0):-copy("NM");
stklayer(0):-copy("METAL");
wirewidth(0):=3;
thespace(0):=metal;
thespace(1):=metal;
thespace(2):=3;
green:=diffusion:=1;
ciflayer(1):-copy("ND");
stklayer(1):-copy("DIFFUSION");
wirewidth(1):=2;
thespace(3):=green;
thespace(4):=green;
thespace(5):=3;
red:=poly:=2;
ciflayer(2):-copy("NP");
stklayer(2):-copy("POLY");
wirewidth(2):=2;
thespace(6):=red;
thespace(7):=red;
thespace(8):=2;

thespace(9):=red;
thespace(10):=green;
thespace(11):=1;

yellow:=implant:=3;
ciflayer(3):-copy("NI");

```

```

glass:=4;
ciflayer(4):-copy("NG");
cuts:=black:=contact:=5;
ciflayer(5):-copy("NC");
buried:=6;
ciflayer(6):-copy("NB");
END of class models;
NEW models(6,11);
lambda:=2.5;
max+lambda+rule:=1.5;
cifscale:=10;
snaptoedge:=true;

```

6.2 An Example

The following program generates a three bit self timed adder chip.

```

BEGIN
EXTERNAL CLASS RNLAP;
RNLAP
BEGIN

REF(primitive) i,j;
REF(point) p;
REF(port) p1,p2;
INTEGER k,l,m;

! 10 reads in an external cell from sticks standard ;

readstick("sadd");
define("bigadd");
  i:-callcell("add").lowerleft;
  addconnector(i.bottom);
  addconnector(i.left);
  addconnector(i.right);
  i:-callref(cell("add")).lowerleft.
  xy(i.cellabut.ul.x,i.cellabut.ul.y);
  addconnector(i.left);
  addconnector(i.right);
  i:-callref(cell("add")).lowerleft.
  xy(i.cellabut.ul.x,i.cellabut.ul.y);
  addconnector(i.left);
  addconnector(i.right);
  addconnector(i.top);
enddef;

! read in the external pads;
readcif("pads");

```

¹⁰The ! is used as a comment flag in SIMULA. Any thing between a ! and ; will be ignored by the compiler.


```

k:=cell("bigadd").left.length;
! k is set to the number of connectors on the left;
define("theins");
  i:-call("padin").rotate(0,1).lowerleft;
  p:-i.connectors.bynam("out");
  j:-be(i.cellabut.ur.x+2.5,p.y);
  layer(diffusion);
  nwire.pnt(p).pnt(j.bynam("g"));
  layer(poly);
  nwire.pnt(j.bynam("r")).dx(3).
  lastisport("out",rightside);
  FOR m:=2 STEP 1 UNTIL k DO
  BEGIN
    ! one for each input;
    i:-callcell("padin").rotate(0,1).
    lowerleft.y(i.cellabut.ul.y);
    p:-i.connectors.bynam("out");
    j:-be(i.cellabut.ur.x+2.5,p.y);
    layer(diffusion);
    nwire.pnt(p).pnt(j.bynam("g"));
    layer(poly);
    nwire.pnt(j.bynam("r")).dx(3).
    lastisport("out",rightside);
  END;
enddef;

!number of connectors on the right;
k:=cell("bigadd").right.length;
define("theouts");
  i:-callcell("padout").rotate(0,-1).lowerright;
  addconnector(i.bottom);
  addconnector(i.connectors.bynam("INPUT")
  .setside(leftside));
  FOR m:=2 STEP 1 UNTIL k DO
  BEGIN ! one for each output;
    i:-callcell("padout").rotate(0,-1).
    lowerright.y(i.cellabut.ul.y);
    addconnector(i.connectors.bynam("INPUT").
    setside(leftside));
  END;
  i:-callcell("padvdd").rotate(0,-1).
  lowerright.y(i.cellabut.ul.y);
  i:-callcell("padgnd").rotate(0,-1).
  lowerright.y(i.cellabut.ul.y);
  addconnector(i.top);
enddef;

define("thechip");
  i:-callcell("theins").lowerleft;
  j:-callcell("bigadd").lowerleft.
  x(i.cellabut.lr.x+50);
  layer(poly);
  riverport(4,2,i.right,j.left);
  i:-callcell("theouts").lowerleft.
  x(j.cellabut.lr.x+50);
  k:=j.right.length;

```

```
    layer(poly);  
    riverport(4,2,j.right,i.left);  
enddef;
```

```
END;  
END of RLAP;
```

7. Utility Programs Written in Rlap

RLAP is extremely useful for writing utilities for processing Stick or CIF files. Because of the internal processing of RLAP the output file will be optimized. All forward references will be removed. In addition CIF numbers will be reassigned for all cells so that two CIF files may be merged.

7.1 CIF to CIF

CIFCIF is a program that takes a CIF file as input and produces an optimized CIF file as output. All forward references in the input CIF file are removed. CIFCIF also can merge several CIF files. To execute the program type the name of the program CIFCIF followed by the output CIF file and a list of input CIF files. The list of input files are one file name followed by another.

The following is the code for CIFCIF. Notice that it is very simple and short.

```
BEGIN
EXTERNAL CLASS RNLAP;
Rnlap
BEGIN
    text t;
    t:-cmdline.getarg;
    while t ne notext do
    begin
        readcif(t);
        t:-cmdline.getarg;
    end;
end;
END of program;
```

7.2 Stick to CIF

STKCIF is a program that takes a Stick file as input and produces an optimized CIF file as output. STKCIF also can merge several Stick files. To execute the program type the name of the program STKCIF followed by the output CIF file and a list of input Stick files. The list of input files are one file name followed by another.

The following is the code for STKCIF. Notice that it is very simple and short.

```
BEGIN
  EXTERNAL CLASS RNLAP;
  Rnlap
  BEGIN
    text t;
    cifscale:=20;
    t:-cmdline.getarg;
    while t ne notext do
      begin
        readstick(t);
        t:-cmdline.getarg;
      end;
    end;
  END of program;
```

References

[Birtwistle 73]

Birtwistle G.M., Dahl O-J., Myhrhaug B. & Nygaard K.
SIMULA begin.
Auerbach Publishers Inc., 1973.

[Kahle 81]

C. Kahle, R.C. Mosteller.
NMOS Sticks Standard Components.
Technical Report SSP File #4300, California Institute of Technology,
1981.

[MEAD 80]

C.A. Mead and L.A. Conway.
Introduction to VLSI Systems.
Addison Wesley, 1980.

[SSP 80]

R.Segal, C.Carroll, G.Tarolli, S.Trimberger, R.Sproull, R.Lyon, D.Lang.
SSP Basic Software Package.
Technical Report SSP MEMO #4024, California Institute of
Technology, 1980.

[Trimberger 80]

Stephen Trimberger.
The Proposed Sticks Standard.
Technical Report SSP MEMO #3487, California Institute of
Technology, 1980.